Using ODS to Generate Excel Files

Chevell Parker

Introduction

This paper will demonstrate techniques on how to effectively generate files that can be read into Microsoft Excel using the Output Delivery System. Topics of discussion will be include the following:

1) Techniques for creating files with ODS that can be read by Excel
2) Common problems that you may encounter along with solutions
3) Advanced techniques using XML and the ODS Markup Language to supply worksheet, workbook, printer, window, data validation, sorting, traffic-lighting and various other properties with ODS
4) Generating Excel file with SAS/INTRNET
5) Example of downloading an Excel file from and HTML page

Some of the tips provided will work with Excel 97, 2000, and XP, but much of what is covered especially, the advanced techniques using XML will apply to Excel 2000 and greater. Many of the techniques used are the CSS style properties. As you will see, creating files with the Output Delivery System that can be read with Excel is very easy, however, some additional work may be needed to customize the output as you like.

Generating Excel files

There are several methods of generating Excel or spreadsheet files using the Output Delivery System. The first method that we will discuss is using the HTML destination to create the Excel or spreadsheet files. When you specify a procedure or data step within the ODS HTML statement with the .XLS or .CSV extensions, Microsoft Excel is opened in the Results Viewer. Excel is not an ODS destination and the fact that the file is opened in Excel is partially a product of ODS. The Results Viewer attempts to open the file based on the files registered extensions. By default the registered extensions of .XLS and .CSV belong to Excel on the PC.

Syntax for creating Excel files with the HTML destination

```
ODS HMTL FILE=”C:\TEMP.XLS”;

PROC PRINT DATA=SASHELP.CLASS;
RUN;

ODS HTML CLOSE;
```

The new ODS CSV destination can also be used to create files that can be opened in Microsoft Excel. The acronym CSV stands for Comma Separated Value. This new destination is experimental with Version 8.2 as part of the ODS Markup Language. The New CSV destination has a tagset which allows the defaults of the destination to be modified as we will see shortly. Excel has the ability to ready CSV files, so specifying the ODS CSV destination with the extension .CSV will create a comma separated file that can be opened in Excel. Also, the delimiter can be changed from a comma to any other delimiter by modifying the CSV tagset. The CSV destination is experimental in Version 8.2 and production for Version 9. Use the CSVALL destination to maintain the titles and footnotes.

Syntax for creating CSV files with ODS

```
ODS CSV FILE="C:\TEMP.CSV";

PROC PRINT DATA=SASHELP.CLASS;
RUN;

ODS CSV CLOSE;
```

ODS HTML vs. ODS CSV

The benefit of using the ODS HTML destination with the .XLS or .CSV extensions is that the formatting is preserved in the Excel file. If you do not use PROC Template to generate a style, the default style is used and carried forward to the Excel file. The drawback to using the ODS HTML destination is that the files tend to be quite large by default which makes the file take longer to load. This is because ODS HTML still uses the 3.2 tagset which allows the formatting and the data to reside in the same file. For example, each cell has a <FONT> tag with all of its attributes along with alignment properties and the like. The HTML 4.0 specification is that the formatting and the data should be separated. The HTML tagsets generated with the ODS Markup Language follows this guideline which is covered in the next section.

Using the ODS CSV destination will generate the traditional spreadsheet file without any formatting. The benefit of using the CSV destination is that because is does not have any formatting, the file size of the CSV file is quite small. The CSV destination does not generate titles and footnotes in the output. The CSVALL destination includes both the titles and footnotes.

Reducing file size

There are a few techniques that can be employed to reduce the size of Excel files and reduce the time it takes for the file to load.

The first thing that we can be done is to create a CSS style sheet with the ODS HTML destination. This allows you to separate the formatting from the data. This reduces the need for each record to have formatting instruction. If you specify the STYLESHEET= option with a file, an external CSS file is generated. If the STYLESHEET option is specified without a file, then the formatting instructions are added to the beginning of the file as an embedded CSS file.

The HTML tagsets can be used to reduce the size of the .XLS files in 8.2 and beyond. With the ODS Markup Language, there are 4 to 5 HTML tagsets that can be used to do this by using the HTML 4.0 standard. The below markup example uses the PHTML tagset to generate the HTML files. This tagset reduces the HTML file the largest while maintaining the formatting. If formatting is not an issue, then you might want to try the CHTML tagset which reduces the file even further, but it does not allow formatting.

The final method for reducing the size of the Excel file is to use the Minimal style. The Minimal style is one of the default styles shipped with SAS. The Minimal style has very few formatting instructions which reduces the size of the file.

ODS HTML Syntax **ODS MARKUP Syntax**

```
ODS HTML FILE=’TEMP.XLS’ STYLESHEET ;      ODS PHTML FILE=’TEMP.XLS’   STYLESHEET=”TEMP.CSS”;

PROC PRINT DATA=SASHELP.CLASS;             PROC PRINT DATA=SASHELP.CLASS;
RUN;                                       RUN;

ODS HTML CLOSE;                            ODS PHTML CLOSE;
```

General Appearance

Titles and Footnotes

Using ODS HTML to create the .XLS or .CSV files will place the entire title or footnote in the first cell. The effect of this is that the first column will become the width of the title or footnote. This occurs because ODS uses the non-standard <Table> tags to house the titles and footnotes which Excel does not expect for a header. This happens for the Bylines as well.

To change this behavior, one of the HTML tagsets can be used. The HTML tagsets all use the header tags <h1> for titles, footnotes and bylines. This is the tag that Excel expects for its headers.

ODS HTML Syntax

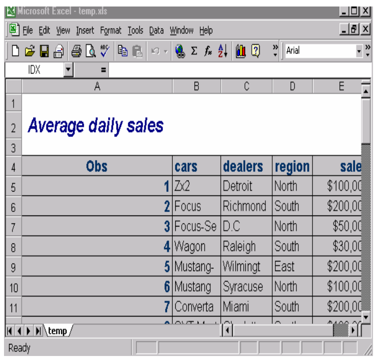
```
ODS HTML FILE=’C:\TEMP.XLS’;

PROC PRINT  DATA=SASHELP.CLASS;
RUN;

ODS HTML CLOSE;
```

ODS MARKUP Syntax

```
ODS HTMLCSS FILE=’C:\TEMP.XLS’ STYLESHEET=”TEMP.CSS”;

PROC PRINT DATA=SASHELP.CLASS;
RUN;

ODS HTMLCSS CLOSE;
```

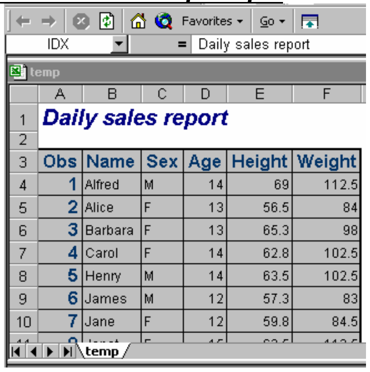
Excel Output

<u>Default HTML Output</u>   ***<u>ODS Markup Output</u>***


Microsoft Excel - temp.xls
File Edit View Insert Format Tools Data Window Help
Arial
IDX
temp
Ready

Average daily sales

Obs	cars	dealers	region	sale
1	Zx2	Detroit	North	$100,00
2	Focus	Richmond	South	$200,00
3	Focus-Se	D.C	North	$50,00
4	Wagon	Raleigh	South	$30,00
5	Mustang-	Wilmingt	East	$200,00
6	Mustang	Syracuse	North	$100,00
7	Converta	Miami	South	$200,00



Daily sales report

Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69	112.5
2	Alice	F	13	56.5	84
3	Barbara	F	13	65.3	98
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5
6	James	M	12	57.3	83
7	Jane	F	12	59.8	84.5

Starting Output in Row 1

HTML

By default when generating .XLS or .CSV files with ODS HTML, the output begins in row 2. This happens because of the non -breaking space character (&nbsp;) in the anchor tag. There is no way to get rid of this anchor tag in the HTML destination other than to post process the HTML file. The HTML tagsets of the Markup Language can be used to start the output in row 1. The HTML tagsets of the ODS Markup Language do not have this non-breaking space character in the anchor tag. See the prior example for syntax.

CSV

The CSV destination generates output beginning in row 3 of the Excel file. This is just the default of the ODS CSV destination. The defaults of the destination can be changed by modifying the CSV tagset and overriding the defaults.. The sample code below modifies the CSV tagset and starts the data in row 1 removing the empty rows.

Tagset syntax to delete empty rows from the CSV destination

```
proc template;
          define tagset tagsets.newcsv;
          parent = tagsets.csv;
            notes "This is the CSV definition";
                  /* we removed the start: put NL. It was putting a line  at the beginning of the table.    */
            define event table;
              finish:
                put NL;
            end;
                  /* We added finish: */
                  /* This makes it so that the line is put at the finish  of the even row instead of every event row.   */
                  define event row;
              finish:
              put NL;
            end;
```

```
  end;
run;

ods tagsets.newcsv body='c:\test.csv' ;
proc print data=sashelp.afmsg label; run;
ods tagsets.newcsv close;
```

Page Setup

Page setup options can be set with a combination of CSS style properties and with XML. In the page set up, we have the ability to modify almost every piece of the page set up such as the margins of the page, the margins of the header and footer, the page orientation, the DPI (data per inch) of the output, the paper size, and pretty much anything else that you want to set.

Setting Margins, Page Orientation, and Page Size

Margins can be set for the page to include the top, bottom, left and right margins. Margins can also be set for headers and footers as well as the vertical and horizontal justification of the page. The paper size, page orientation and the first page number can all be set.

The margins for the page can be set using the CSS style property Margin. The margins for the headers and footers can be specified using the Microsoft Office specific **mso-header-margin** and **mso-footer-margin**. The alignment of the table horizontally and vertically on the page can be set using the **mso-horizontal-page-align** and the **mso-vertical-page-align** CSS style properties. The page orientation can be set within the @Page rule by using the style property **mso-page-orientation** with the possible values of portrait or landscape. To set the page orientation, in addition to specifying this style property, the <ValidPrintInfo> tag would have to be specified within the Print XML element.(See example on page 20). The paper size can be modified using the <PaperSizeIndex> XML tag within the Print element. To set the first page number, use the **mso-first-numbers** style property.

Syntax for setting those items included in the page setup

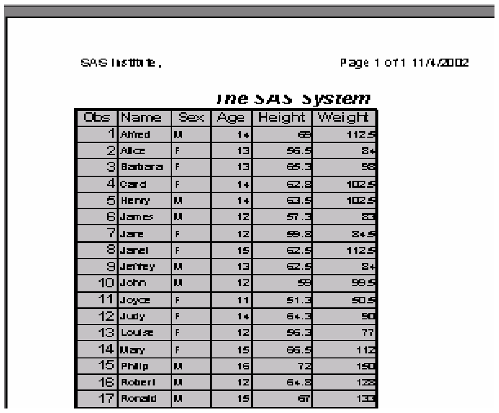
```
ods html file='temp.xls' headtext=
'<style> @page { margin:1.0in .75in 1.0in .75in;
                 mso-header-margin:.5in;
                 mso-footer-margin:.5in;
                 mso-horizontal-page-align:center;
                 mso-vertical-page-align:center;
                 mso-first-numbers:1  }
</style>';

proc print data=sashelp.class;
run;

ods html close;
```

Setting Headers and Footers

Headers and footers can also be defined within the @Page rule using the CSS style properties **mso-header-data** and **mso-footer-data**. This allows you to specify customized headers for the printed output. The headers and footers can be a generic page number, page X of Y format, date time, a signature, customized on the left, right, top and bottom that include a variety of the fore-mentioned. The below example uses the Page X of Y header at the top of the page and some customized text at the left, center, right and the bottom of the page. The HEADTEXT= option is limited to 256 characters including the spaces. If the string goes over this limit, it will be truncated and none of the attribute will be displayed because it truncates the closing </style> tag. If you need more than the 256, it is a good idea to add this information in the doc_head section of the Tagset. For examples of doing this, see the example section in the references.

Syntax for adding headers and footers

```
ods htmlcss file='temp.xls'  stylesheet="temp.css" headtext=
'<style>
   @Page
      {mso-header-data:”&L SAS INSTITUTE &C&Page &P of &N”;
        mso-footer-data:"&Lthis is the left&CPage &P&RThis is the
          right&A&D&T&N&P"};
</style>';

proc print data=sashelp.class;
run;

ods htmlcss close;
```



SAS Institute.   Page 1 of 1 11/4/2002

The SAS System

Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69	112.5
2	Alice	F	13	56.5	84
3	Barbara	F	13	65.3	98
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5
6	James	M	12	57.3	83
7	Jane	F	12	59.8	84.5
8	Janet	F	15	62.5	112.5
9	Jeffrey	M	13	62.5	84
10	John	M	12	59	99.5
11	Joyce	F	11	51.3	50.5
12	Judy	F	14	64.3	90
13	Louise	F	12	56.3	77
14	Mary	F	15	66.5	112
15	Philip	M	16	72	150
16	Robert	M	12	64.8	128
17	Ronald	M	15	67	133

Cell Formatting

If you ever attempted to import a HTML file with Excel, then you are aware of the problems that are associated with doing this. One of the largest problems is with cell formatting. The problems that are encountered are no different than when using ODS HTML to generate a .CSV or .XLS file. The problem occurs because Excel uses a general format to import cell values. The general format reads the cell values as they are typed, however, there are some common problems that you should be aware of.

Common Problems

- Both numeric and character variables will lose leading zeroes when creating .XLS or .CSV files with the ODS Destination. You will not realize the problem until the leading zeroes are omitted from an account number, an Id, or a zip code.

- Numeric variables with a length greater than 11 digits end up in scientific notation.
- Numeric variables with a dash separating values are interpreted as Excel dates

- Trailing zeroes in the decimal values are lost also when the cell values are imported into Excel.

- The height and width specified for cell values within SAS are ignored when a .XLS or .CSV file is created with ODS.
- Output forced to a new line using the <BR> tag is read as a separate observation

Excel Dates

- Dates are stored in Excel as serial numbers from Jan 1, 1900, or if the option is set Jan 1, 1904. To change to the 1904 date system, click Options on the Tools menu, click the Calculation tab, and then select the 1904 date system check box. This can also be changed within SAS using XML which will be shown later

- Consideration has to be given when importing non formatted dates

- Excel handles formatted dates correctly using the general format

The following table shows the first date and the last date for each date system and the serial value associated with each date.

Date
system
First date
Last date
1900 January 1, 1900
(serial value 1) December 31, 9999
(serial value 2958465)

1904 January 2, 1904
(serial value 1) December 31, 9999
(serial value 2957003**)**

Formatting Solutions

There are various ways of getting around some of the cell formatting issues which we will discuss in the section below. Most of the solutions will use the Microsoft Office specific CSS style sheet properties to format the cell values.

Importing values as Text

Importing the file as text using the Text format for the cell values allow the cell values to come over without any interpretation and does not strip the leading or trailing zeroes. Using **the mso-number-format:\@** CSS style property allows the cell value to be imported using the Text format for Excel 2000 and above. For Excel 97, the CSS style property is **vnd.ms-excel.numberformat**. We can specify any of the many format understood by Excel, or just format the cells as text.

Common Cell Formats

`mso-number-format:0`	NO Decimals
`mso-number-format:"0\.000"`	3 Decimals
`mso-number-format:"\#\,\#\#0\.000"`	Thousand comma with 3 decimals
`mso-number-format:"mm\/dd\/yy"`	Date7
`mso-number-format:"mmmm\\ d\\\,\\ yyyy"`	Date9
`mso-number-format:"m\/d\/yy\\ h\:mm\\ AM\/PM"`	Date-time AMPM
`mso-number-format:"Medium Date"`	01-mar-98
`mso-number-format:"d\\-mmm\\-yyyy"`	01-mar-1998
`mso-number-format:"Short Time"`	5:16
`mso-number-format:"Medium Time"`	5:16 am
`mso-number-format:"Long Time"`	5:16:21:00
`mso-number-format:Percent`	Percent
`mso-number-format:0%`	No percent
`mso-number-format:"0\.E+00"`	Fractions
`mso-number-format:"\@"`	Text
`mso-number-format:"\[<=9999999\]\#\#\#\\-\#\#\#\#\;\\\(\#\#\#\\\)\\ \#\#\#\\-\#\#\#\#"`	Phone number

Applying Cell Formats

Excel 2000 Solution

Text Format applied to all columns

```
ODS HTML FILE=‘TEMP.HTML’ HEADTEXT=
“<STYLE> TD {MSO-NUMBER-FORMAT:\@}</STYLE>”;

PROC PRINT DATA=SASHELP.CLASS;
RUN;

ODS HTML CLOSE;
```

Text Format applied to the age field

```
ODS HTML FILE=”TEMP.HTML HEADTEXT=
‘<Style> .Zero { mso-number-format:\@;}</Style>’;

PROC PRINT DATA=SASHELP.CLASS;
 VAR  AGE /  style={HTMLCLASS=”Zero”};
RUN;

ODS  HTML CLOSE;
```

Excel 97 solution

```
DATA ONE;
  X='0001';
RUN;

ODS HTML FILE='TEMP.XLS';

PROC PRINT DATA=ONE;
VAR X / STYLE={HTMLSTYLE="VND.MS-EXCEL.NUMBERFORMAT:@"};
RUN;

ODS HTML CLOSE;
```

Applying Cell Formats in the CSV destination

To prevent losing the leading zeroes when using the CSV destination, an “=” can by added in front of the character string. This allows the fields to be read with the text format. This workaround also works with the HTML destination. The CSV tagset can also be modified to add the “=” before the data values.

Syntax to prevent losing the leading zeroes in the CSV destination

```
/* Syntax to alter data value by pre-pending the “=”.  */

data one;
    x='="00001"';
run;

ods csv file='tempx.csv';

  proc print data=one;
  run;

ods csv close;

/* PROC TEMPLATE code that modifies the default CSV tagset  by  appending the ‘=’  */

proc template;
   define tagset Tagsets.test;
      parent=tagsets.csv;

      define event data;
         put "," / if !cmp( COLSTART , "1" );
         put '=' """" / if cmp( TYPE , "string" );
         put VALUE;
         put """" / if cmp( TYPE , "string" );
      end;
```

```
 end;
run;


 data one;
    x='00001';
 run;

ods markup file='temp.csv' tagset=tagsets.test;

proc print data=one;
run;
ods markup close;
```

Specifying the Height and Width

When the height and width are set in SAS, they may be ignored by Microsoft Excel. If you set the height or the width, this adds the WIDTH= or HEIGHT= attributes to the HTML TD tag. If you are creating CSS style sheets it adds the WIDTH and HEIGHT CSS style properties. By default, the width of the cell will be the length of the largest value of the cell either in the browser or within the spreadsheet. To override this behavior, you can use the CELLWIDTH= or the CELLHEIGHT= style attributes to modify the width or the height.

This all works fine when going to the browser, however, Excel is pretty picky. If you are using the ODS HTML destination in SAS 8.2 or 9.0, specifying titles, footnotes, or using By variables which creates bylines will cause Excel to ignore any height or width set. This is because title, footnotes, and the by-lines are stored in HTML tables within the ODS HTML destination. The first table in the spreadsheets sets the column width and the height for the cells. Since the titles, footnotes and by-lines are specified before all other tables, all of the height and width information in the subsequent tables are negated. There are ways around this problem which I will discuss below.

1. If you do not need titles in the output , you can use a null title statement and remove the titles from the output. This will prevent the cell width and the cell height from getting negated.

   ```
   TITLE;
   ```

2. Using the ODS MARKUP destination provides the other solution. Neither the HTMLCSS or the PHTML tagsets put the titles in HTML tables, therefore we do not run into the same problem that we do with the HTML destination. However, using tagsets has it own problem in that if you do not specify a unit of dimension, it automatically specifies .px(pixels). This does not make Excel very happy when a unit of dimension is specified with the Width= HTML attribute. To get around this, we are able to specify the WIDTH CSS style parameter which allows you to specify a unit of measure in Excel. The PROC TEMPLATE code is used to add the HTMLSTYLE to the data event.

Syntax to specify cell width

```
proc template;
 define tagset tagsets.width;
  parent=tagsets.htmlcss;
      define event data;
         put "<td";
            trigger pre_post;
            putq " class=" HTMLCLASS;
            trigger align;
            trigger style_inline;
            trigger rowcol;
            put ' style=' htmlstyle;
            put ">";
            put "<pre>" / if exist( asis );
            put VALUE;
         finish:
            trigger pre_post;
            put "</pre>" / if exist( asis );
            put "</td>" NL;
      end;
end;
run;
ods markup tagset=tagsets.width  path=”c:\”(url=none) file='markup.xls' stylesheet='css.css';
proc print data=sashelp.class;
var age / style={htmlstyle="width:200pt"}
var name  sex height weight;
run;
ods markup close;
```

Preventing <BR> tag from generating separate cells

Excel creates separate cells for values that contain the <BR> HTML tag. The <BR> HTML tag is responsible for generating a new line. This is often used to split lengthy cell values in into multiple lines. To change this behavior, the **mso-data-placement:same-cell** CSS property and value can be used to prevent Excel from creating a separate cell when the <BR> HTML tag is used. When you specify attributes such as the SPLIT= option in PROC REPORT and you have such things as the Orientation set, or the page titles set, they are simply ignored if you do not this is not set. Not sure of the reason, but this is the solution. Below is an example.

Syntax to prevent the <BR> tag from creating a new cell

```
ODS HTML FILE=’TEMP.HTML’
HEADTEXT=”<STYLE> BR {mso-data-placement:same-cell}</STYLE>”;

PROC  PRINT DATA=SASHELP.CLASS;
RUN;

ODS HTML CLOSE;
```

Generating Customized Borders

Generating customized borders can be done within SAS using PROC TEMPLATE or with procedures that support the STYLE= option. There are times when you might not want borders around all of the cells, but still some of the cells. This section will show you how to generate customized borders at the cell level. The first thing that we will do is to turn off the borders at the table level so we can customize the borders of individual cells. To do this   use the CSS style property Border. The Border style attribute has 3 separate values: weight, style, and color. The border style property can be used with the style attribute HTMLSTYLE to control the borders on an individual level.

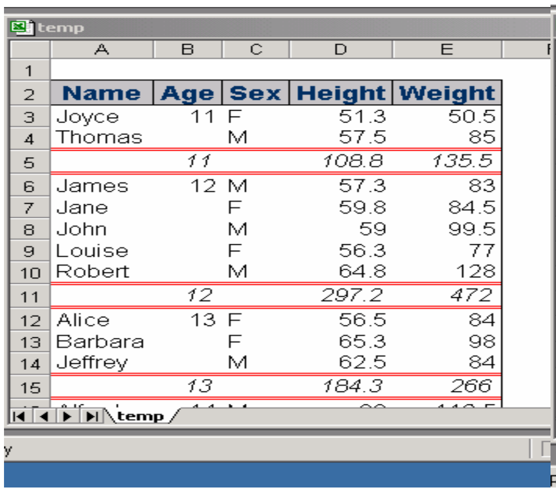
The border style property will control the overall border with the border-left, border-right, border-top and border-bottom style properties controlling the various parts of the border. Below are some examples that show how this is done. For more information on this, visit the PROC TEMPLATE FAQ.

Creates a double red underline beneath the sub-totals that is 5px in width

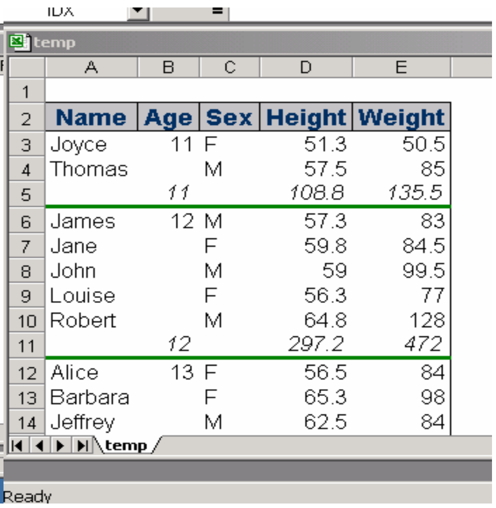
```
ODS HTML FILE='TEMP.XLS' ;

PROC REPORT DATA=SASHELP.CLASS NOWD STYLE(REPORT)={Rules=none } style(column)=
{background=white htmlstyle='border:none'} ;
COL Name Age Sex Height Weight;
Define age / order;

BREAK AFTER AGE  / SUMMARIZE STYLE={HTMLSTYLE="border-bottom:5px  double red;border-left:none;border-
right:none;BORDER-TOP:5px double red"};
RUN;
ODS HTML CLOSE;
```

Creates a single green bottom border

```
ODS HTML FILE='TEMP.XLS' ;

PROC REPORT DATA=SASHELP.CLASS NOWD STYLE(REPORT)={Rules=none } style(column)=
{background=white htmlstyle='border:none'} ;
COL Name Age Sex Height Weight;
Define age / order;

BREAK AFTER AGE  / SUMMARIZE STYLE={HTMLSTYLE="border-bottom:5px solid green;border-left:none;border-
right:none;BORDER-TOP:none"};
RUN;
ODS HTML CLOSE;
```



	A	B	C	D	E
1					
2	Name	Age	Sex	Height	Weight
3	Joyce	11	F	51.3	50.5
4	Thomas		M	57.5	85
5		11		108.8	135.5
6	James	12	M	57.3	83
7	Jane		F	59.8	84.5
8	John		M	59	99.5
9	Louise		F	56.3	77
10	Robert		M	64.8	128
11		12		297.2	472
12	Alice	13	F	56.5	84
13	Barbara		F	65.3	98
14	Jeffrey		M	62.5	84
15		13		184.3	266



	A	B	C	D	E
1					
2	Name	Age	Sex	Height	Weight
3	Joyce	11	F	51.3	50.5
4	Thomas		M	57.5	85
5		11		108.8	135.5
6	James	12	M	57.3	83
7	Jane		F	59.8	84.5
8	John		M	59	99.5
9	Louise		F	56.3	77
10	Robert		M	64.8	128
11		12		297.2	472
12	Alice	13	F	56.5	84
13	Barbara		F	65.3	98
14	Jeffrey		M	62.5	84

Modifying Colors, Fonts and Alignments

Style attributes such as colors, fonts and alignments can be modified with PROC TEMPLATE. For procedures that support the STYLE= option such as PRINT, REPORT, and TABULATE, the style attributes can be set within the procedure. Excel will honor most of the style information that the browser will honor. Some exceptions are the alignment for HTML tables and titles with the HTML destination. HTML tables are left justified by default within Excel. Titles and footnotes are stored within HTML tables in 8.2, therefore titles and footnotes are also left by default.

When using the ODS Markup Language, most of the style information such as colors, fonts and alignments are not generated in the HTML or Excel file by default. The Markup Language follows the HTML 4.0 standard, which separates the data from the formatting by using a CSS style sheets for the formatting. Excel 2000 and later recognizes CSS Style Sheets, while Excel 97 does not. If you want to preserve the formatting while using Excel 97, then the HTML destination is the best option. When using PROC TEMPLATE and creating a CSS style sheet with the STYLESHEET= option, modifications made in the template are translated into the CSS file. Below is PROC TEMPLATE code to modify the colors and fonts of the .XLS file.

PROC TEMPLATE Syntax to modify various parts of the output

```
PROC TEMPLATE;
  DEFINE STYLE STYLES.TEST;
     PARENT=STYLES.DEFAULT;
        STYLE Header FROM Header /                /* Responsible for column headers   */
              FONT_SIZE=6;
        STYLE Data FROM Data /                    /*  Responsible for cell values      */
              BACKGROUND=WHITE;
        STYLE SystemTitle FROM SystemTitle /      /* Responsible for the titles        */
              FOREGROUND=RED;
```

```
      STYLE SystemFooter FROM SystemFooter  /        /* Responsible for the footnotes      */
  END;
RUN;

ODS HTML FILE='TEMP.HTML' STYLE=STYLES.TEST;
PROC PRINT DATA=SASHELP.CLASS;
RUN;
ODS HTML CLOSE;
```

Creating Multiple Worksheets within a Workbook

Multiple worksheets can be generated within a workbook with ODS by using XML to supply the various worksheets which is covered in the next section. This can also be done with a macro that uses a combination of DDE and ODS HTML. This macro is documented in the following location. This can also be done strictly using DDE.
http://support.sas.com/techsup/unotes/SN/006/006695.html

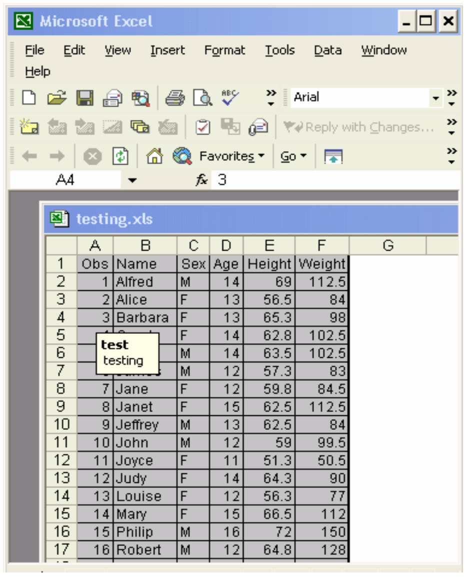
Using XML to Modify Excel Options

XML can be used to modify options and supply information to Excel from SAS. With the use of XML and he CSS style properties, we can pretty much modify every part of the Excel file from SAS. We will just touch on the power on which XML plays in modifying the .XLS files.

With the use of XML, we can do everything from modify the resolution of the printed output, selecting the number of copies to the naming the worksheets within the workbook, to scaling the output, generating backups, splitting windows, modifying the window size, data validation, sorting, conditional formatting, setting formulas, setting filters, supplying and removing gridlines, protecting cells, supplying or removing scroll bars, target screen size and the list continues indefinitely. Much of the XML is supplied as an XML data island within the header section of the HTML file. I hope to show some examples of the power of XML when using the HTML destination. For more information on XML and Excel, please see the XML and HTML resource guide in the references. Also see the gallery of examples which are also located in resources.

1) The first example uses XML to validate data that is passed to Excel.
   For the cell A4, the value has to be a whole number which is specified with the Type node. If the value is not an integer, the field will be flagged. Also the maximum value of cell a 4 can be 100 and a minimum value of 1. This is specified by the Max and Min nodes. If the value does not conform, a error is generated when the file is imported into Excel. Also when the cell is active, a title and error message is generated by specifying the InputTitle Node and the InputMessage nodes. You can also specify error messages for the cell when the value is invalid. If you have a title statement, Excel generates a warning.

```
proc template;
  define tagset tagsets.test;
    parent=tagsets.phtml;
        define event doc;
         start:
            put HTMLDOCTYPE NL NL NL;
            put '<html xmlns:o="urn:schemas-microsoft-com:office:office"' NL;
            put 'xmlns:x="urn:schemas-microsoft-com:office:excel"' NL;
            put 'xmlns="http://www.w3.org/TR/REC-html40">' NL;

         finish:
            put "</html>" NL;
        end;
        define event doc_head;
         start:
            put "<head>" NL;
            put VALUE NL;
            put "<style>" NL "<!--" NL;
            trigger alignstyle;
            put "-->" NL "</style>" NL;
            put " <!--[if gte mso 9]><xml>" NL;
            put "<x:DataValidation>" NL;
            put "  <x:Range>A4</x:Range>" NL;
            put "   <x:Type>Whole</x:Type>" NL;
            put "   <x:Min>1</x:Min>" NL;
            put "   <x:Max>1</x:Max>" NL;
            put "   <x:InputTitle>test</x:InputTitle>" NL;
            put "   <x:InputMessage>testing</x:InputMessage>" NL;
            put "</x:DataValidation>" NL;
            put "</xml><![endif]-->"  NL;
         finish:
            put "</head>" NL;
          end;
    end;
run;
options center;
ods markup file="c:\testing.html" tagset=tagsets.test stylesheet='c:\temp.css';

proc print data=sashelp.class;
title;
run;

ods markup close;
```



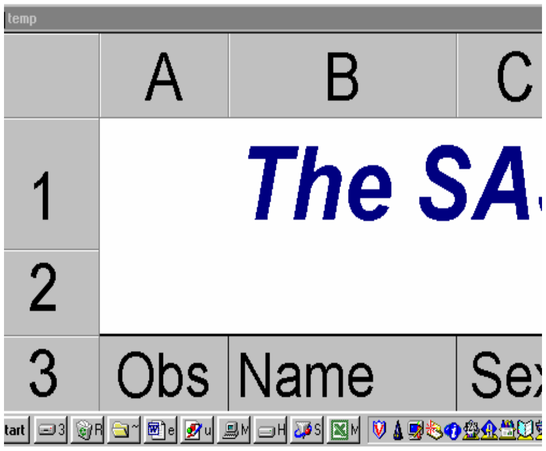
	A	B	C	D	E	F	G
1	Obs	Name	Sex	Age	Height	Weight	
2	1	Alfred	M	14	69	112.5	
3	2	Alice	F	13	56.5	84	
4	3	Barbara	F	13	65.3	98	
5	[illegible]	[illegible]	F	14	62.8	102.5	
6	[illegible]	[illegible]	M	14	63.5	102.5	
7	[illegible]	[illegible]	M	12	57.3	83	
8	7	Jane	F	12	59.8	84.5	
9	8	Janet	F	15	62.5	112.5	
10	9	Jeffrey	M	13	62.5	84	
11	10	John	M	12	59	99.5	
12	11	Joyce	F	11	51.3	50.5	
13	12	Judy	F	14	64.3	90	
14	13	Louise	F	12	56.3	77	
15	14	Mary	F	15	66.5	112	
16	15	Philip	M	16	72	150	
17	16	Robert	M	12	64.8	128	

test
testing

2) The below example creates a workbook by the name of temp and a single worksheet with the name sheet1. As with the previous example, the Name node is specified within the ExcelWorksheets node to name the worksheet. Within the WorksheetsOption there is the Zoom node. Within the Zoom node we have specified that the output should scale 400% of its actual size. We also specify that cell 1 is the active cell. The window width and height and protection levels are also set.

```
 proc template;
   define tagset tagsets.test;
     parent=tagsets.phtml;
        define event doc;
         start:
            put HTMLDOCTYPE NL NL NL;
            put '<html xmlns:o="urn:schemas-microsoft-com:office:office"' NL;
            put 'xmlns:x="urn:schemas-microsoft-com:office:excel"' NL;
            put 'xmlns="http://www.w3.org/TR/REC-html40">' NL;

         finish:
            put "</html>" NL;
        end;

        define event doc_head;
         start:
            put "<head>" NL;
            put VALUE NL;
            put "<style>" NL "<!--" NL;
            trigger alignstyle;
            put "-->" NL "</style>" NL;
         finish:

       put "  <!--[if gte mso 9]><xml>" NL;
       put "  <x:ExcelWorkbook>" NL;
       put "   <x:ExcelWorksheets>" NL;
       put "    <x:ExcelWorksheet>" NL;
       put "     <x:Name>Sheet1</x:Name>" NL;
       put "     <x:WorksheetOptions>" NL;
       put "      <x:Zoom>400</x:Zoom>" NL;
       put "      <x:Selected/>" NL;
       put "      <x:Panes>" NL;
       put "       <x:Pane>" NL;
       put "       <x:Number>3</x:Number>" NL;
       put "      <x:ActiveCol>1</x:ActiveCol>" NL;
       put "      </x:Pane>" NL;
       put "    </x:Panes>" NL;
       put "   <x:ProtectContents>False</x:ProtectContents>" NL;
       put "  <x:ProtectObjects>False</x:ProtectObjects>" NL;
       put "  <x:ProtectScenarios>False</x:ProtectScenarios>" NL;
       put "  </x:WorksheetOptions>" NL;
       put "  </x:ExcelWorksheet> " NL;
       put "  <x:WindowHeight>8070</x:WindowHeight> " NL;
       put "  <x:WindowWidth>10380</x:WindowWidth> " NL;
       put "  <x:WindowTopX>480</x:WindowTopX> " NL;
       put "  <x:WindowTopY>120</x:WindowTopY> " NL;
       put "  <x:ProtectStructure>False</x:ProtectStructure> " NL;
       put "  <x:ProtectWindows>False</x:ProtectWindows>" NL;
       put "  </x:ExcelWorkbook>" NL;
       put "  </xml><![endif]--> " NL;
     put "</head>" NL;
         end;
    end;
run;
options center;
```

```
ods markup file="c:\temp.xls" tagset=tagsets.test stylesheet='c:\temp.css';
proc print data=sashelp.class;
run;

ods markup close;
```



3) The below example generates multiple worksheets for a workbook by specifying HTML files to import into this workbook. Each existing HTML file will become a spreadsheet in this workbook. Within the ExcelWorksheet nodes, I have named 3 separate sheet names which are first, second, and third. Within the WorksheetSource node I specify the URL for the specific sheet. The HTML files have to already exist before plugging it into this tag. This example creates a workbook with the name xx.xls and the three worksheets that were mentioned above. Version 9.1 offers a more dynamic way of doing this using the ExcelXP tagset which generates the multiple worksheets dynamically making use of the XML Spreadsheet format. This also requires that you have Excel 2002.

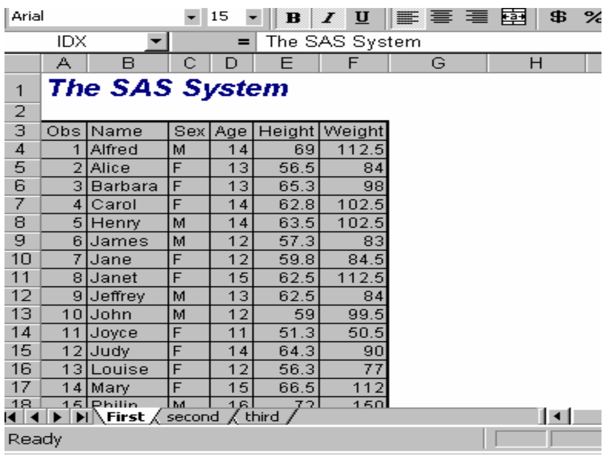
   When the workbook is opened, a dialog box is displayed which informs you that all of the files are not in the location that Excel is expecting them. This is because we don't store the files completely as Excel would. There are a few ways around this error message.

   1) Saving the file as a native XLS file rather than the HTML format which is it is Currently. If you need to do this programmatically then we can use DDE if you are on the PC. See the example page in the references for the exact code.

   2) Emulate the file structure that Excel expects by default. You will see an explanation and code in the examples section which you will find in the references.

```
/*  Sample HTML files to include into the
  proc sort data=sashelp.class out=test;
  by age;
  run;

ods html file="c:\temp.html" newfile=bygroup;
```

```
proc report data=test nowd;
by age;
run;

ods html close;


  proc template;
   define tagset tagsets.test;
     parent=tagsets.phtml;

      define event doc;
        start:
            put HTMLDOCTYPE NL NL NL;
            put '<html xmlns:o="urn:schemas-microsoft-com:office:office"' NL;
            put 'xmlns:x="urn:schemas-microsoft-com:office:excel"' NL;
            put 'xmlns="http://www.w3.org/TR/REC-html40">' NL;
        finish:
            put "</html>" NL;
        end;

        define event doc_head;
         start:
            put "<head>" NL;
            put  ' <meta name="Excel Workbook Frameset">';
            put  '<meta http-equiv=Content-Type content="text/html; charset=windows-1252">';
            put  '<meta name=ProgId content=Excel.Sheet>';
            put  '<meta name=Generator content="Microsoft Excel 9">';

        finish:

        put "<!--[if gte mso 9]><xml>" NL;
        put "<x:ExcelWorkbook>" NL;
        put "  <x:ExcelWorksheets>" NL;
        put "   <x:ExcelWorksheet>" NL;
        put "    <x:Name>first</x:Name>" NL;
        put "    <x:WorksheetSource HRef='c:\temp.html'/>" NL;
        put "  </x:ExcelWorksheet>" NL;
        put "   <x:ExcelWorksheet>" NL;
        put "     <x:Name>second</x:Name>" NL;
        put "     <x:WorksheetSource HRef='c:\temp1.html'/>" NL;
        put "  </x:ExcelWorksheet>" NL;
        put "  <x:ExcelWorksheet> " NL;
        put "     <x:Name>third</x:Name>" NL;
        put "     <x:WorksheetSource HRef='C:\temp2.html'/>" NL;
        put "  </x:ExcelWorksheet>" NL;
        put "   </x:ExcelWorksheets>" NL;
        put " <x:WindowHeight>8070</x:WindowHeight>" NL;
        put " <x:WindowWidth>10380</x:WindowWidth>" NL;
        put "<x:WindowTopX>480</x:WindowTopX>" NL;
        put "<x:WindowTopY>45</x:WindowTopY>" NL;
        put "<x:ActiveSheet>3</x:ActiveSheet>" NL;
        put "<x:ProtectStructure>False</x:ProtectStructure>" NL;
        put "<x:ProtectWindows>False</x:ProtectWindows>" NL;
        put "</x:ExcelWorkbook>" NL;
        put "</xml><![endif]-->" NL;
        put "</head>" NL;
          end;
    end;
run;

ods markup file="c:\xx.xls" tagset=tagsets.test stylesheet='c:\temp.css';

data _null_;
file print;
   put;
run;

ods markup close;
```

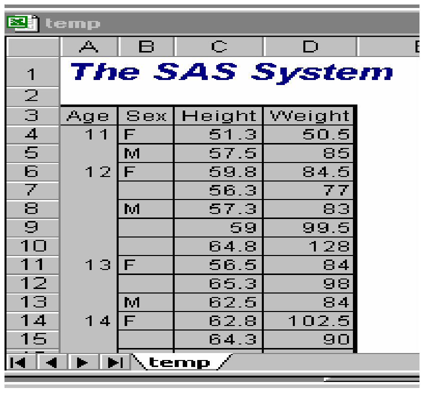


The SAS System

	Obs	Name	Sex	Age	Height	Weight
4	1	Alfred	M	14	69	112.5
5	2	Alice	F	13	56.5	84
6	3	Barbara	F	13	65.3	98
7	4	Carol	F	14	62.8	102.5
8	5	Henry	M	14	63.5	102.5
9	6	James	M	12	57.3	83
10	7	Jane	F	12	59.8	84.5
11	8	Janet	F	15	62.5	112.5
12	9	Jeffrey	M	13	62.5	84
13	10	John	M	12	59	99.5
14	11	Joyce	F	11	51.3	50.5
15	12	Judy	F	14	64.3	90
16	13	Louise	F	12	56.3	77
17	14	Mary	F	15	66.5	112
18	15	Philip	M	16	72	150

4) Style information can be applied conditionally using the ConditionalFormatting node. With the Range Node, the cells are specified that will take a certain action based on the condition. If the cell A4 is not empty, then cells A1-A21 will not have a border around its cells.

```
proc template;
   define tagset tagsets.test;
     parent=tagsets.phtml;
       define event doc;
start:
    put HTMLDOCTYPE NL NL NL;
    put '<html xmlns:o="urn:schemas-microsoft-com:office:office"' NL;
    put 'xmlns:x="urn:schemas-microsoft-com:office:excel"' NL;
    put 'xmlns="http://www.w3.org/TR/REC-html40">' NL;
 finish:
     put "</html>" NL;
  end;
    define event doc_head;
        start:
            put "<head>" NL;
            put VALUE NL;
            put "<style>" NL "<!--" NL;
            trigger alignstyle;
            put "-->" NL "</style>" NL;
        finish:
          put "<!--[if gte mso 9]><xml>" nL;
          put "<x:ConditionalFormatting>" NL;
          put "  <x:Range>A4:A21</x:Range>" NL;
          put  "  <x:Condition>" NL;
          put  '   <x:Value1>$A$4<>""</x:Value1>' NL;
          put  "    <x:Format Style='border:none;'/>" NL;
          put "   <x:Condition>" NL;
          put "</x:ConditionFormatting>" NL;
          put "</xml><![endif]-->" NL;
          put "</head>" NL;
          end;
    end;
run;
options center;
ods markup file="c:\temp.xls" tagset=tagsets.test stylesheet='c:\temp.css';

proc report data=sashelp.class nowd;
col age sex height weight;
define age / order;
define  sex / order;
run;

ods markup close;
```



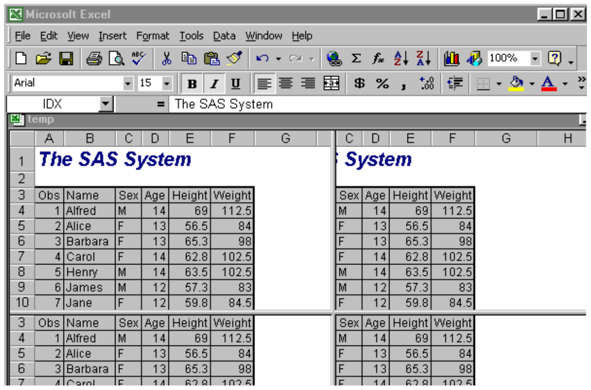
The SAS System

Age	Sex	Height	Weight
11	F	51.3	50.5
	M	57.5	85
12	F	59.8	84.5
		56.3	77
	M	57.3	83
		59	99.5
		64.8	128
13	F	56.5	84
		65.3	98
	M	62.5	84
14	F	62.8	102.5
		64.3	90

temp

5) The final example shows a combination of supplying printer options with XML, worksheet options, and workbook options. The Print node specifies that the printed output is scaled at 85% of the actual output by specifying this argument within the Scale node. The horizontal and vertical resolution is specified using the HorizontalResultion and VerticalResolution nodes with an argument of 300 DPI. This value might save your printer ribbon wear. We then specify that the widow should be split using the SplitHorizontal and the SplitVertical nodes and that the cells should not be protected in the Excel file.

```
proc template;
  define tagset tagsets.test;
    parent=tagsets.phtml;
define event doc;
start:
   put HTMLDOCTYPE NL NL NL;
   put '<html xmlns:o="urn:schemas-microsoft-com:office:office"' NL;
   put 'xmlns:x="urn:schemas-microsoft-com:office:excel"' NL;
   put 'xmlns="http://www.w3.org/TR/REC-html40">' NL;
 finish:
    put "</html>" NL;
 end;

      define event doc_head;
       start:
         put "<head>" NL;
         put VALUE NL;
         put "<style>" NL "<!--" NL;
         trigger alignstyle;
         put "-->" NL "</style>" NL;

       finish:

put  "<!--[if gte mso 9]><xml>" nL;
put  " <x:ExcelWorkbook>" NL;
put  "  <x:ExcelWorksheets>" NL;
put  "   <x:ExcelWorksheet>" NL;
put  "    <x:Name>xxx</x:Name>" NL;
put  "   <x:WorksheetOptions>" NL;
put  "    <x:Print>" NL;
put  "    <x:ValidPrinterInfo/>" NL;
put  "    <x:PaperSizeIndex>5</x:PaperSizeIndex>" NL;
put  "     <x:Scale>85</x:Scale>" NL;
put "      <x:HorizontalResolution>300</x:HorizontalResolution>" NL;
put "      <x:VerticalResolution>300</x:VerticalResolution>" NL;
put "     </x:Print>" NL;
put "    <x:Selected/>" NL;
put "     <x:DoNotDisplayGridlines/>" NL;
put "     <x:SplitHorizontal>2805</x:SplitHorizontal>" NL;
put "     <x:TopRowBottomPane>2</x:TopRowBottomPane>" NL;
put "     <x:SplitVertical>4950</x:SplitVertical>" NL;
put "     <x:LeftColumnRightPane>2</x:LeftColumnRightPane>" NL;
put "    <x:Panes>" NL;
put "     <x:Pane>" NL;
put "      <x:Number>3</x:Number>" NL;
put "      </x:Pane>" NL;
put "     <x:Pane>" NL;
put "       <x:Number>1</x:Number>" NL;
      put  " <x:ActiveCol>5</x:ActiveCol>" NL;
      put  "</x:Pane>" NL;
      put  "<x:Pane>" NL;
      put  "<x:Number>2</x:Number>" NL;
      put  " <x:ActiveRow>14</x:ActiveRow>" NL;
      put  "<x:ActiveCol>4</x:ActiveCol>" NL;
      put  "</x:Pane>" NL;
      put  "<x:Pane>" NL;
      put  " <x:Number>0</x:Number>" NL;
      put  " <x:ActiveRow>14</x:ActiveRow>" NL;
      put  " <x:ActiveCol>7</x:ActiveCol>" NL;
      put  "</x:Pane>" NL;
      put  "</x:Panes>" NL;
      put  " <x:ProtectContents>False</x:ProtectContents>" NL;
      put  " <x:ProtectObjects>False</x:ProtectObjects>" NL;
      put  "<x:ProtectScenarios>False</x:ProtectScenarios>" NL;
      put  "</x:WorksheetOptions>" NL;
      put " </x:ExcelWorksheet>" NL;
      put  " </x:ExcelWorksheets>" NL;
      put  " <x:WindowHeight>6030</x:WindowHeight>" NL;
      put  "<x:WindowWidth>10380</x:WindowWidth>" NL;
      put  "<x:WindowTopX>480</x:WindowTopX>" NL;
      put  " <x:WindowTopY>135</x:WindowTopY>" NL;
      put  "<x:ProtectStructure>False</x:ProtectStructure>" nL;
      put  "<x:ProtectWindows>False</x:ProtectWindows>" NL;
      put  "</x:ExcelWorkbook>" nL;
      put  "</xml><![endif]-->" NL;
      put "</head>" NL;
         end;
   end;
run;
options center;
ods markup file="c:\temp.xls" tagset=tagsets.test stylesheet='c:\temp.css';

proc print data=sashelp.class;
run;

ods markup close;
```



= The SAS System

The SAS System

Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69	112.5
2	Alice	F	13	56.5	84
3	Barbara	F	13	65.3	98
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5
6	James	M	12	57.3	83
7	Jane	F	12	59.8	84.5

System

Sex	Age	Height	Weight
M	14	69	112.5
F	13	56.5	84
F	13	65.3	98
F	14	62.8	102.5
M	14	63.5	102.5
M	12	57.3	83
F	12	59.8	84.5

Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69	112.5
2	Alice	F	13	56.5	84
3	Barbara	F	13	65.3	98

Sex	Age	Height	Weight
M	14	69	112.5
F	13	56.5	84
F	13	65.3	98

6) The below example generates landscape output in the .XLS file created. Within the WorksheetOptions element and further the Print element is the ValidPrinterInfo tag that is needed to change the orientation to landscape. In addition, the mso-page-orientation attribute has to be set. In the below example the mso-page-orientation style attribute is set in the HEADTEXT= ODS option.

```
proc template;
 define tagset tagsets.test;
  parent=tagsets.htmlcss;
    define event doc;
    start:
     put '<html xmlns:o="urn:schemas-microsoft-com:office: office"' NL;
     put 'xmlns:x="urn:schemas-microsoft-com:office:excel">'  NL;
   finish:
   put "</html>" NL;
  end;
 define event doc_head;
 start:
   put "<head>" NL;
   put VALUE NL;
   put "<style>" NL;
   put "<!--" NL;
   trigger alignstyle;
   put "-->" NL;
   put  "</style>" NL;
 finish:
   put "<!--[if gte mso 9]><xml>" NL;
   put "<x:ExcelWorkbook>" NL;
   put "<x:ExcelWorksheets>" NL;
   put "   <x:ExcelWorksheet>" NL;
   put "    <x:Name>Sheet1</x:Name>" NL;
   put "    <x:WorksheetOptions>" NL;
   put "    <x:DisplayPageBreak/>" NL;
   put "     <x:Print>" NL;
   put "      <x:ValidPrinterInfo/>" NL;
   put "     </x:Print>" NL;
   put "    </x:WorksheetOptions>" NL;
   put "   </x:ExcelWorksheet>" NL;
   put "  </x:ExcelWorksheets>" NL;
   put "</x:ExcelWorkbook>" NL;
   put "</xml><![endif]-->" NL;
   put "</head>" NL;
 end;
end;
run;

ods markup file="c:\temp.html" tagset=tagsets.test
stylesheet='c:\temp.css'
headtext="<style> @page {mso-page-orientation:landscape;}</style>" ;
   proc print data=sashelp.class;
   title;
   run;

ods markup close;
```

Generating Excel files with SAS/INTRNET

To generate Excel files with SAS/INTRNET, the appropriate content-type has to be set for the file. This can be done either with the data step or beginning in Version 8.1 using the APPSRV_HEADER function with the appropriate content-type. Below is an example.

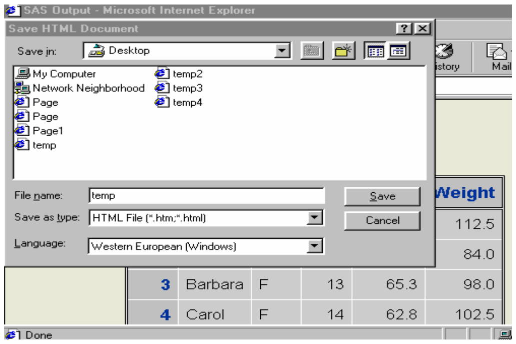
```
%LET RV=%SYSFUNC(APPSRV_HEADER(CONTENT-TYPE, APPLICATION/VND.MS-EXCEL));
%LET RV=%SYSFUNC(APPSRV_HEADER(CONTENT-DISPOSITION, %STR(ATTACHMENT; FILENAME=TEMP.XLS)));

 ODS HTML BODY=_WEBOUT ;
 PROC PRINT DATA=SASHELP.CLASS;
 RUN;
 ODS HTML CLOSE;
```

Downloading Excel files from a HTML page

Files can be saved as Excel files from an HTML page by adding a button that issues the document.ExecCommand function with the saveas argument. This opens the dialog window with the name of the file as the file to download. To download this file as an Excel file, just add the extension of .XLS or .CSV and the Excel version of this HTML file will be saved. If you would like to name the file before the dialog box is opened with the filename path and extension, you can specify it as an argument to the command. The path specified will need to be specified with “\\” slashes in the path where there is a single “\”. Also note within the SAVEAS command is two single quotes. If you do not name the file as demonstrated in the first example, then

```
/*  Syntax  which names file */
proc template;
  define style styles.test;
     parent=styles.default;
      style body from body /
          prehtml='<input onclick="document.execCommand(‘’SAVEAS’’,true,"c:\\temp\\test.xls")"
          value="Save As" type="button">';
end;
run;

ods html file='temp.html' style=styles.test;
proc print data=sashelp.class;
run;
ods html close;


/* Syntax  which uses default */

proc template;
  define style styles.test;
     parent=styles.default;
                 style body from body /
```

```
                        prehtml='<input onclick="document.execCommand(‘’saveas’’)"
                           value="Save As" type="button">';
   end;
run;

ods html file='temp.html' style=styles.test;

proc print data=sashelp.class;
run;

ods html close;
```



Save in: Desktop

My Computer
Network Neighborhood
Page
Page
Page1
temp
temp2
temp3
temp4

File name: temp

Save as type: HTML File (*.htm;*.html)

Language: Western European (Windows)

					Weight
					112.5
					84.0
3	Barbara	F	13	65.3	98.0
4	Carol	F	14	62.8	102.5

Conclusion

The Output Delivery System is just one of the ways within SAS to create files that can be read by Excel. What I attempted to do was to show the common problems associated with creating these files using ODS. There are so many more items that could have been addressed here, but we had to end somewhere.

References

Microsoft® Office HTML and XML Reference
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnoffxml/html/ofxml2k.asp

“Generating Custom Excel Files Using ODS”
http://www2.sas.com/proceedings/sugi28/012-28.pdf

Sample Gallery of Programs
http://support.sas.com/rnd/base/topics/templateFAQ/Excelsamples.html

Contact Information

You can contact me personally at Chevell.Parker@sas.com. Also visit the Base R&D web site which is located at : http://support.sas.com/rnd/base/for a host of other useful ODS and BASE SAS topics which is updated pretty frequently.